Multi-tenant-SaaS – Performance

Paper: Eddy Truyen, André Jacobs, Stef Verreydt, Emad Heydari Beni, Bert Lagaisse, and Wouter Joosen. 2020. Feasibility of container orchestration for adaptive performance isolation in multi-tenant SaaS applications. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (*SAC '20*). Association for Computing Machinery, New York, NY, USA, 162–169. DOI: https://doi.org/10.1145/3341105.3374034

Benchmark: k8-scalar, https://github.com/k8-scalar/k8-scalar/blob/master/docs/tutorial.md

- K8-Scalar experiment-component https://github.com/k8-scalar/k8-scalar/tree/master/studies/WOC2019/Experiments/Operations/experiment-controller
    - **Do not use the stress.sh script but configure a scalar workload profile directly**: https://github.com/k8-scalar/k8-scalar/blob/master/docs/scalar/features.md#configuration-of-workload-profile
- Multi-tenant SaaS: https://github.com/k8-scalar/k8-scalar/tree/master/studies/WOC2019/Experiments/Operations/saas-app/kube_deployment

Testbed: Minikube

Scope in terms of relevant Kubernetes features:

- Container networking
    - hostNetwork setting for a Pod
- Application configuration and deployment
    - Deployments
    - Services
- Resource quota management
    - Namespaces
- Container QoS management
    - Requests and Limits
    - LimitRange
    - QoS classes
    - CPU management policies
    - Topology management policies

Problems to address:

- The paper proposes an architecture for QoS differentiation between different SLA classes of tenants, but not against aggressive tenants that flood the application with a too high request rate. After all, tenants that belong to the same SLA class are hosted by the same set of Pods. Therefore:
    - Try to build a stronger performance isolation mechanism against aggressive tenants, even if this comes at a higher resource cost.
    - Does performance isolation improve when applying CPU management policies or topology management policies?
- Run the aggressive SaaS Pods as a host privileged Pod that has access to the host network. Does this allow the aggressive tenant to have a bigger impact?

Tips

- Not all features of this assignment have been covered in great details during the lectures. Present to your fellow students a deeper study of these features and to which extent they are feasible to address the problems
- Deploy two instances of the experiment-controller on different nodes of the cluster. One experiment controller represents a good tenant that adheres to the maximum request rate, while the other experiment controller is an aggressive tenant that tries to take the service down with a denial of service attack. Run both experiment-controller in parallel. How is the performance affected in terms of average latency per peak phase
- Deploy now two golden SLA services where each tenant is served by a separate service. Repeat the above experiment with the good and aggressive tenant. How is the performance in terms of average latency?
- To simulate multiple tenants, create multiple ExperimentController StatefulSets, do not scale the replicas of a single StatefulSets.

Continuous deployment and In-place updates of Pod resources – Performance

Important: since the group consists of 3 members, this project consists of two coherent parts. For both parts the same paper is to be read.

Paper: Eddy Truyen, Arnout Hoebreckx, Cédric De Dycker, Bert Lagaisse and Wouter Joosen 2020. Flexible Migration in Blue-Green Deployments within a Fixed Cost. In *Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds* (*WOC'20*). Association for Computing Machinery, New York, NY, USA, 13–18. DOI: https://doi.org/10.1145/3429885.3429963

Part 1

Benchmark: k8-scalar, https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/docs/tutorial.md

- K8-scalar experiment-component:
  - https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/studies/WOC2020/experiments/0-experiment-sharedlibraries
  - https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/studies/WOC2020/experiments/experiment-2-Latency/experiment-execution
- Helm charts
  - Multi-tenant SaaS version 1: https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/studies/WOC2020/experiments/0-experiment-setup/mt-api-v1
  - Multi-tenant SaaS version 2: https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/studies/WOC2020/experiments/0-experiment-setup/mt-api-v2
- Upgradeplanner:
  - Development: https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/studies/WOC2020/upgradeplanner/upgradeplanner
  - Operations: https://github.com/k8-scalar/k8-scalar/tree/compatible-k8-1.14/studies/WOC2020/experiments/0-experiment-setup/upgradeplanner/upgradedeployment.yml

Testbed: Minikube v1.24 and Helm v 1.28: https://github.com/k8-scalar/k8-scalar/blob/compatible-k8-1.14/docs/tutorial.md#1-setup-a-kubernetes-cluster-helm-and-install-the-heapster-monitoring-service

Scope in terms of relevant Kubernetes features:

- Application configuration and deployment
  - Blue-Green Deployments
  - Customization of the rolling upgrade strategy
- Container QoS Management
  - Requests and Limits
  - Vertical scaling
  - Node Allocatable
  - Cgroup structure of Pods

Problems to address:

- The paper presents an analytical model that allows to calculate the highest amount of tenants that can be consolidated on a single node so that on-line migration of tenants in blue-green deployments without any service disruption will not run into a deadlock due to due to insufficient node resources.
  - Apply the analytical model to your minikube testbed to determine the maximum number of tenants that can be hosted. Reproduce the latency experiment by running experiment-2-Latency/experiment-execution/exp-automated-group.sh. This experiment will exercise a peak load during an upgrade. What is the impact of the latency in comparison to a normal load by running experiment-1-Usage/experiment-execution/exp-automated-group.sh
- The motivation of this paper makes the case that it pays off to co-locate different versions of the same Pod on the same node in order to safe memory. After all, memory can be saved when the Pods have dynamically linked libraries in common. This motivation is supported by reproducing the results of the paper's reference [14] for the nginx application.
  - Testbed: As above
  - Reproduce the 0-experiment-sharedlibraries experiment for the npm application that is deployed in the mt-api-v1 and mt-api-2 deployments. Generate similar computations as calculated by Table 1 in the paper by using the "pmap –XX <Pid>" command in "su mode" on your minikube VM. How much

memory is saved in comparison to a situation where mt-api-v1 and mt-api-v2 Pods would have been deployed in separate nodes. Try to calculate what these memory savings might mean for the analytical model. Do you think an additional tenant could be hosted in your minikube testbed if the cgroup structure of the Pod would have been re-implemented as proposed in the paper

- Apply on paper (not in experiment!) the analytical model to calculate the highest amount of tenants that can be hosted on your minikube node when opting out for Pod restarts during vertical scaling

Part 2

Benchmark: k8-scalar, https://github.com/k8-scalar/k8-scalar/blob/master/docs/tutorial.md

- K8-Scalar experiment-component https://github.com/k8-scalar/k8-scalar/tree/master/studies/WOC2019/Experiments/Operations/experiment-controller
    - **Do not use the stress.sh script but configure a scalar workload profile directly**: https://github.com/k8-scalar/k8-scalar/blob/master/docs/scalar/features.md#configuration-of-workload-profile
    - Multi-tenant SaaS: https://github.com/k8-scalar/k8-scalar/tree/master/studies/WOC2019/Experiments/Operations/saas-app/kube_deployment

Testbed:

- Install the **latest** minikube version with support for K8s version 1.28.
- **Activate the feature gate `InPlacePodVerticalScaling` for in-place update of Pod resources. This makes it possible to update resource request and limit settings of running Pods without restarting Pods.**

Problem to address:

- To support vertical scaling without pod restarts, the reliance on the `InPlacePodVerticalScaling` feature is needed. Assess the reliability of the feature:
    - Modify the `deploy_saas_kube.yaml` file of the multi-tenant SaaS application:
        - Add CPU and Memory requests and limits to the deployment yaml file
        - Enable in place updates of CPU and memory without pod restarts by adding appropriate configuration settings to the yaml file
    - Next deploy the SaaS application via kubectl commands
        - Deploy the application using `kubectl create -f`
        - Expose the service of the application using `kubectl create -f`
    - Now update the running application by editing the Deployment resource and updating cpu and memory request and limits that fit within the minikube's node resources.
        - Do you need to modify the deployment to prevent the rolling upgrade process that normally would take place without the feature gate activate?
    - Now update the running application by editing the Deployment resource and updating cpu and memory requests that will exceed the minikube's node resources.
        - What do you think that will happen?
        - What did happen?

General tips for both parts:

- Not all k8s features of this assignment have been covered in great details during the lectures. Present to your fellow students a deeper study of these features and to which extent they are feasible to address the problems
- The analytical model assumes that vertical scaling of Pods, e.g. changing the requests or limits of an existing Pod, is used for migrating resources from an older version to a new version. Remember that vertical scaling of a Pod normally requires replacing that Pod with a new one and therefore service of the Pod is temporary disrupted without additional measures. More specifically, a rolling upgrade with maximum surge and 100% availability is used. As a side-effect, during the upgrade a higher number of Pods will need to run on the node. This is termed surge cost in the paper and taken into account by the analytical model.
- The analytical model proposed the concept of unit of resource for serving a single tenant. This is specified under the metadata section of each deployment as separate request and limit labels. These labels should also be defined for memory
- Understanding the output of pmap:
    - https://johanlouwers.blogspot.com/2017/07/oracle-linux-understanding-linux.html
    - https://lwn.net/Articles/230975/
    - Understanding the output of pmap